Available online at www.sciencedirect.com

SCIENCE DIRECT®

Theoretical Computer Science 337 (2005) 119–133

Theoretical Computer Science

www.elsevier.com/locate/tcs

The generative capacity of block-synchronized context-free grammars☆

I. McQuillan*

Department of Computer Science, University of Western Ontario, Middlesex College 383, London, Ont., Canada, N6A 5B7

Received 26 February 2003; received in revised form 19 February 2004; accepted 2 November 2004

Communicated by O.H. Ibarra

Abstract

We consider the yield languages of synchronized tree automata, called the synchronized context-free (SCF) languages. We show that their language family coincides with the family of ET0L languages using both studied types of synchronization. Furthermore, we examine a generalization of SCF grammars, the block-synchronized context-free (BSCF) grammars and determine that their generated language family is equal to that of the indexed languages using the same two types of synchronization. However, when the nesting depth of BSCF grammars is bounded above by some constant, the generated language family is also equal to the family of ET0L languages. This shows that the unbounded nesting depth language family is strictly larger than the bounded nesting depth family, as previously conjectured.

Keywords: Synchronization; ET0L languages; Rewriting systems; Indexed languages

1. Introduction

The study of *synchronization* was originally initiated by Hromkovič et al. (see [7–9]) to model the communication between parallel computations of Turing Machines and

☆ This research was funded in part by the Natural Sciences and Engineering Research Council of Canada through Grant OGP0000243 (H. Jürgensen).

* Corresponding author. Tel.: +1 519 661 3566; fax: +1 519 661 3515.
E-mail address: imcquill@csd.uwo.ca (I. McQuillan).

0304-3975/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2004.11.003

alternating machines. Salomaa then introduced *synchronized tree automata* which are top down tree recognizers. These allow some communication between parallel computations along different paths of the trees (see [14]). Jürgensen and Salomaa then created *synchronized context-free grammars* (or SCF grammars) to study the string languages generated by nondeterministic synchronized tree automata (see [10]). With this model, each nonterminal may or may not use a *synchronization symbol* which allow different branches to communicate. There are various ways in which branches can communicate. In particular, two methods, equality and prefix synchronization were studied in [10] and their respective language families were shown to be equal. Intuitively, with prefix synchronization, if a branch uses a synchronization symbol, it must wait until all other branches use the same synchronization symbol, or terminate, before continuing. Equality synchronization is similar, however every branch that uses a synchronization symbol *must* wait until all other branches use the same symbol. In addition, a generalization of SCF grammars was introduced which enabled each branch to recursively start a subderivation, whereby synchronization occurs between branches only within the same subderivation. This system was called *weak derivation, block-synchronized context-free grammars* (or BSCF grammars). We will characterize the language families generated by SCF and BSCF grammars using both types of synchronization to other well-known, important language families including the ET0L languages (see [13]) and the indexed languages (see [1]).

2. Preliminaries

An alphabet A is a finite, nonempty set of symbols. The set of all words over A is denoted by A^*, and this set contains the empty word, λ. A prefix of a word w is any y such that $w = yx$ for some word x which we denote by $y \leqslant_{\mathrm{p}} w$. Also, $w_1 \simeq_{\mathrm{p}} w_2$ if and only if one of w_1 or w_2 is a prefix of the other. We call $\mathbb{N}$ and $\mathbb{N}_0$ the set of positive and nonnegative integers, respectively.

A *context-free grammar* is denoted $G = (V, T, I, P)$, where V and T are disjoint alphabets of nonterminals and terminals, respectively, $I \in V$ is the starting nonterminal, and P is a finite set of productions of the form $X \to w$ where $X \in V$ and $w \in (V \cup T)^*$. Instead of using the usual rewrite relation, derivations of context-free grammars can be represented as trees.

A *tree domain* D is a nonempty finite subset of $\mathbb{N}^*$ such that

(1) If $\mu \in D$, then every prefix of μ belongs to D.
(2) For every $\mu \in D$ there exists $i \geqslant 0$ such that $\mu j \in D$ if and only if $1 \leqslant j \leqslant i$.

Let A be a set. An *A-labelled tree* is a mapping $t : D \to A$, where D is a tree domain. Elements of D are called nodes of the tree and D is said to be the domain of t, $\mathrm{dom}(t)$. A node $\mu \in \mathrm{dom}(t)$ is labelled by $t(\mu)$. A node $\lambda \in \mathrm{dom}(t)$ is called the root of t. The set of leaves of t is denoted $\mathrm{leaf}(t)$. The subtree of t at node μ is t/μ. When there is no confusion, we refer to a node also simply by its label.

Nodes of a tree t that are not leaves are called *inner nodes* of t. The *inner tree of* t, $\mathrm{inner}(t)$ is the tree obtained from t by cutting off all the leaves. The *yield* of an A-labelled tree t, $\mathrm{yd}(t)$, is the word obtained by concatenating the labels of the leaves of t from left to right; the leaves are ordered by the lexicographic ordering of $\mathbb{N}^*$. For $\mu \in \mathrm{dom}(t)$, $\mathrm{path}_t(\mu)$ is the sequence of symbols of A occurring on the path from the root of t to the node μ.

Let $G = (V, T, I, P)$ be a CF grammar. A $(V \cup T \cup \{\lambda\})$-labelled tree t is a *derivation tree of* G if it satisfies the following conditions.

(1) The root of t is labelled by the initial nonterminal, that is, $t(\lambda) = I$.
(2) The leaves of t are labelled by terminals or by the symbol λ.
(3) Let $\mu \in \mathrm{dom}(t)$ have k immediate successors, $k \geqslant 1$. Then $t(\mu) \to t(\mu 1) \cdots t(\mu k) \in P$.

The set of derivation trees of G is denoted $T(G)$. The derivation trees of G are in one-to-one correspondence with the equivalence classes of derivations of G producing terminal words, and thus

$$L(G) = \{\mathrm{yd}(t) \mid t \in T(G)\}. \tag{1}$$

Above, in the word $\mathrm{yd}(t)$, we identify occurrences of the symbol λ with the empty word. The family of context-free languages is denoted $\mathcal{L}(\mathrm{CF})$ (see [6] for an introduction).

An *ET0L system* is denoted $G = (V, V_1, \beta, w)$, where V is a finite set of nonterminals, V_1 is a nonempty subset of V which are the terminals, $w \in V^+$ is the starting word, and β is a finite set of systems of productions, $\beta = \{P_1, P_2, \ldots P_r\}$. Each P_i is a finite set of productions of the form $a \to w_a$ where $a \in V$, and $w_a \in V^*$; there must be at least one production for each nonterminal in each P_i. The rewrite relation $\Rightarrow_G$ is defined by the condition that $x \Rightarrow_G y$ if and only if $x = a_{i_1} \cdots a_{i_m}$, $y = y_1 y_2 \cdots y_m$, and there is a table $P \in \beta$ such that $a_{i_j} \to y_j \in P$ for $1 \leqslant j \leqslant m$. Let $\Rightarrow_G^*$ be the transitive and reflexive closure of the relation $\Rightarrow_G$. The language generated by G is defined by $L(G) = \{x \mid x \in V_1^*, w \Rightarrow_G^* x\}$. The family of languages generated by ET0L systems is denoted $\mathcal{L}(\mathrm{ET0L})$ (see [13,4]).

An *indexed grammar* is denoted $G = (V, T, F, P, I)$, where V is a finite set of nonterminals, T is a finite set of terminals disjoint from V, $I \in V$ is the initial nonterminal, $F = \{f_1, \ldots, f_r\}$ is the set of indices, where, for $1 \leqslant j \leqslant r$, f_j is a finite set of productions of the form $X \to w$ with $X \in V, w \in V^*$, and P is a finite set of productions of the form $X \to w$ with $X \in V, w \in (VF^* \cup T)^*$. The rewrite relation $\Rightarrow_G$ is defined by the conditions that $x \Rightarrow_G y$ if and only if either one of the following two cases hold:

(1) $x = w_1 X \beta w_2$ for some $w_1, w_2 \in (VF^* \cup T)^*$, $X \in V$, $\beta \in F^*$,
$X \to C_1\beta_1 C_2\beta_2 \cdots C_m\beta_m \in P$ and $y = w_1 C_1\gamma_1 C_2\gamma_2 \cdots C_m\gamma_m w_2$, $\beta_j \in F^*$,
where $\gamma_j = \beta_j\beta$ if $C_j \in V$ and $\gamma_j = \lambda$ if $C_j \in T$, $1 \leqslant j \leqslant m$,
(2) $x = w_1 X f \beta w_2$ for some $w_1, w_2 \in (VF^* \cup T)^*$, $X \in V$, $f \in F$, $\beta \in F^*$,
$X \to C_1 C_2 \cdots C_m \in f$ and $y = w_1 C_1\gamma_1 C_2\gamma_2 \cdots C_m\gamma_m w_2$,
where $\gamma_j = \beta$ if $C_j \in V$ and $\gamma_j = \lambda$ if $C_j \in T$, $1 \leqslant j \leqslant m$.

Let $\Rightarrow_G^*$ be the transitive and reflexive closure of the relation $\Rightarrow_G$. The language generated by G is defined by $L(G) = \{w \mid w \in T^*, I \Rightarrow_G^* w\}$. The family of languages generated by indexed grammars is denoted by $\mathcal{L}(\mathrm{IND})$.

An indexed grammar is in *normal form* if and only if the following conditions hold:

(1) for all $f \in F$, each production in f has form $X \to Y$ with $X, Y \in V$,
(2) each production of P is either of the form $X \to YZ$, $X \to Yf$, $X \to a$ or $X \to \lambda$ where $X, Y, Z \in V$, $f \in F$, $a \in T$.

We can assume that an indexed language is generated by an indexed grammar in normal form without loss of generality (see [1]).

3. Block-synchronization grammars

We will now give some definitions to describe block-synchronization context-free grammars. Although lengthy, the intended intuition is quite simple. For clarification, see Example 10 below. We refer to [10] for additional definitions and examples.

A (*block*) *synchronization alphabet* is a finite set

$$\Xi = S \cup B \cup \{e\}, \tag{2}$$

where S is the set of *situation symbols*, B is the set of *begin-symbols*, and e is the *end-symbol*, all pairwise disjoint.

The set of *well-formed synchronizing sequences* (or *sync-sequences*, for short) over the alphabet Ξ, $\mathrm{WF}(\Xi)$, is defined as being the smallest subset of Ξ^* satisfying the condition:

$$u_0 b_1 v_1 e u_1 b_2 v_2 e u_2 b_3 \dots u_{m-1} b_m v_m e u_m \in \mathrm{WF}(\Xi), \tag{3}$$

for every $v_1, \dots, v_m \in \mathrm{WF}(\Xi)$, $b_1, \dots, b_m \in B$, $u_0, \dots, u_m \in S^*$, $m \geqslant 0$.

In the following, let w be a well-formed synchronizing sequence as in (3) for some $v_1, \dots, v_m \in \mathrm{WF}(\Xi)$, $b_1, \dots, b_m \in B$, $u_0, \dots, u_m \in S^*$.

The *nesting depth* of w, $\mathrm{nd}(w)$, is defined by

$$\mathrm{nd}(w) = \begin{cases} 0 & \text{if } m = 0, \\ 1 + \max\{\mathrm{nd}(v_i) \mid i = 1, \dots, m\} & \text{if } m \geqslant 1. \end{cases} \tag{4}$$

Let $w \in \mathrm{WF}(\Xi)$ be as in (3). The 0-level *situation sequence* of w is $\mathrm{sit}(w) = u_0 \cdots u_m$. Thus, it is the word obtained by removing all symbols which are parenthesized.

Let Ξ be a synchronization alphabet and $w_1, w_2 \in \mathrm{WF}(\Xi)$,

Definition 1. w_1, w_2 are said to be p-*similar* (prefix-similar), $w_1 \sim_{\mathrm{p}} w_2$ if $\mathrm{sit}(w_1) \simeq_{\mathrm{p}} \mathrm{sit}(w_2)$. w_1, w_2 are said to be e-*similar* (equality-similar), $w_1 \sim_{\mathrm{e}} w_2$ if $\mathrm{sit}(w_1) = \mathrm{sit}(w_2)$.

The sequences w_1 and w_2 are p-similar if their 0-level situation sequences are in prefix-relation. Next, we define a BSCF grammar. When referring to G as a BSCF grammar, one means that the derivations of G are restricted by the synchronization conditions defined below in Definition 6.

Let Ξ be a synchronization alphabet.

Definition 2. A *block-synchronized context-free grammar* (BSCF grammar), is a context-free grammar

$$G = (N, T, I, P), \tag{5}$$

where $N = V \times (\Xi \cup \{\lambda\})$. We will denote $G = (V, \Xi, T, I, P)$ in the sequel.

Nonterminals of $V \times \Xi$ are called the *synchronizing nonterminals* of G. Nonterminals of the form $V \times \{\lambda\}$, are called *nonsynchronizing nonterminals*. Elements of V are the *base nonterminals*.

The grammar G is a *synchronized context-free grammar*, *SCF* grammar, if in (5) Ξ is replaced by a set of situation symbols S.

We define the morphism $h_G : (V \times (\Xi \cup \{\lambda\}))^* \longrightarrow \Xi^*$ by the condition $h_G((v, x)) = x$ for all $v \in V$ and $x \in \Xi \cup \{\lambda\}$.

Definition 3. Let G be a BSCF grammar. A derivation tree t of G is said to be *well-formed* if for every leaf v of inner(t), the word $h_G(\text{path}_t(v))$ is a well-formed synchronizing sequence.

Definition 4. Let $G = (V, \Xi, T, I, P)$ be a BSCF grammar and let $t \in T(G)$ be well-formed. We say that $\mu \in \text{dom}(t)$ is a *begin-node* (respectively, *end-node*) if μ is labelled by a begin-nonterminal (respectively, end-nonterminal) of G.

We say that an end-node v *corresponds* to a begin-node μ if $h_G(\text{path}_{t/\mu}(v))$ is a well-formed sync-sequence. If μ is a begin-node of t, we denote the set of end-nodes corresponding to μ by $\text{CORR}_t(\mu)$.

The *blocktree* of t corresponding to a begin-node μ, $\text{block}_t(\mu)$, is the tree that is obtained from inner$(t)/\mu$ by deleting all subtrees below the nodes of $\text{CORR}_t(\mu)$.

This implies that every blocktree is a well-formed tree.

Definition 5. Let G be a BSCF grammar and t be a well-formed derivation tree of G.

(1) Let $t_1 = \text{inner}(t)$ and let $\mu \in \text{leaf}(t_1)$. The synchronizing *sequence* (sync-sequence) corresponding to μ is $\text{seq}_{t_1}(\mu) = h_G(\text{path}_{t_1}(\mu))$.
(2) Let t_2 be the blocktree corresponding to some node labelled by a begin-nonterminal of t and let $\mu \in \text{leaf}(t_2)$. The synchronizing *sequence* of t_2 corresponding to the leaf μ, $\text{seq}_{t_2}(\mu)$, is $h_G(\text{path}_{t_2}(\mu)) = b \cdot \text{seq}_{t_2}(\mu) \cdot e$ for some begin symbol b.

Next, we will restrict the trees that will be used to generate BSCF languages.

Definition 6. Let $G = (V, \Xi, T, I, P)$ be a BSCF grammar and $z \in \{\text{p}, \text{e}\}$. A well-formed derivation tree t of G is said to be z-acceptable if the following holds always when t_1 is a blocktree of t or $t_1 = \text{inner}(t)$.

$$\text{For all } \mu, \nu \in \text{leaf}(t_1), \text{seq}_{t_1}(\mu) \sim_z \text{seq}_{t_1}(\nu). \tag{6}$$

Let $z \in \{\text{p}, \text{e}\}$ and G be a BSCF grammar. The set of z-acceptable derivation trees of G is denoted $T_z(G)$.

Definition 7. For $z \in \{\text{p}, \text{e}\}$, the language *z-synchronized generated* by G is $L_z(G) = \text{yd}(T_z(G))$.

A language L is a *z-BSCF language*, $z \in \{\text{p}, \text{e}\}$, if there exists a BSCF grammar G such that $L = L_z(G)$.

The families of z-BSCF languages are denoted by $\mathcal{L}_z(\text{BSCF})$. L is a z-SCF language, $z \in \{\text{p}, \text{e}\}$, if there exists an SCF grammar G such that $L = \text{yd}(T_z(G))$, and the corresponding language family is denoted $\mathcal{L}_z(\text{SCF})$.

We refer to BSCF grammars and languages as weak derivation BSCF grammars and languages to distinguish between the strong derivation type also introduced in [10]. With

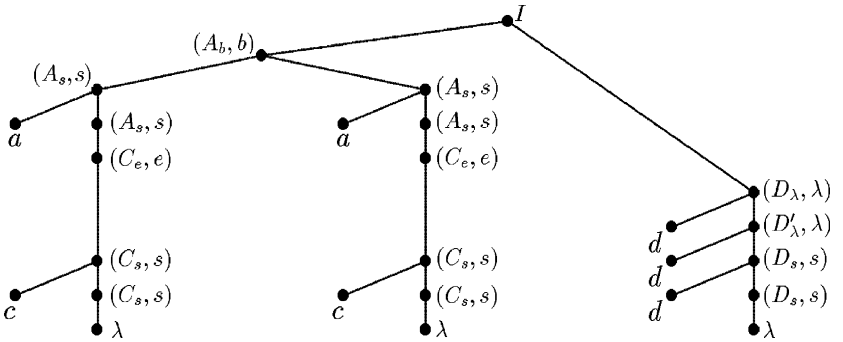

Fig. 1. A derivation tree $t \in T_{\mathrm{e}}(G)$.

strong derivations, synchronization occurs as above, however, branches in distinct blocktrees must also synchronize if they are created directly inside the same blocktree.

Definition 8. Let t be a well-formed derivation tree of G, v a begin-node and denote $t_1 = \mathrm{block}_t(v)$. The *depth* of the blocktree t_1 is defined to be

$$\max\{\mathrm{nd}(\mathrm{seq}_{t_1}(\mu)) \mid \mu \in \mathrm{leaf}(t_1)\}. \tag{7}$$

Definition 9. Let G be a BSCF grammar and $c \in \mathbb{N}_0$. We say that $t \in T_z(G)$ has *nesting depth* c if all sync-sequences $\mathrm{seq}_{\mathrm{inner}(t)}(\mu)$, $\mu \in \mathrm{leaf}(\mathrm{inner}(t))$, have depth at most c. The family of z-BSCF languages of finite nesting depth is denoted $\mathcal{L}_z(\mathrm{fnBSCF})$.

Example 10. Define $G = (V, \Xi, T, I, P)$, $\Xi = S \cup B \cup \{e\}$, where $S = \{s\}$, $B = \{b\}$, $V = \{A_b, A_s, C_e, C_s, D_\lambda, D'_\lambda, D_s, I\}$, $T = \{a, c, d\}$ and P contains the following productions:

$I \to (A_b, b)(D_\lambda, \lambda)$, $(A_b, b) \to (A_s, s)(A_s, s)$,
$(A_s, s) \to a(A_s, s) \mid (C_e, e)$, $(C_e, e) \to (C_s, s)$,
$(C_s, s) \to c(C_s, s) \mid \lambda$, $(D_\lambda, \lambda) \to d(D'_\lambda, \lambda)$,
$(D'_\lambda, \lambda) \to d(D_s, s)$, $(D_s, s) \to d(D_s, s) \mid \lambda$.

Consider the tree $t \in T_{\mathrm{e}}(G)$ in Fig. 1. The first two nonterminals of the rightmost branch need not synchronize with any other branches since they are nonsynchronizing nonterminals. The first branch starts a blocktree. Within the blocktree, the branches synchronize, but they do not with branches outside the blocktree. Thus, the number of times s is used as situation symbol must be the same for each branch in the blocktree and it must be the same for each branch outside the blocktree. Hence, $\mathrm{yd}(t) = acacd^3$ and $L_{\mathrm{e}}(G) = \{a^n c^m a^n c^m d^{m+2} \mid n, m \geqslant 0\}$.

4. Finite nesting depth weak derivations

First, we provide a normal form for BSCF grammars which states that every BSCF language can be defined by a grammar which has a synchronizing symbol in every nonterminal.

Consequently, using nonsynchronizing nonterminals to hide events does not increase the generative power, but they can lead to simplified grammars and proofs.

Informally, we create a new situation symbol, denoted \$, which acts in a similar fashion to the empty word. If there is a production with nonsynchronizing nonterminals on the right-hand side, then for those nonsynchronizing nonterminals, the empty word is replaced by \$. For all the synchronized nonterminals on the right-hand side, we replace the situation symbol with \$, and then only allow these new nonterminals to loop, or go to the original synchronized nonterminal. Also, all occurrences of nonsynchronizing nonterminals are replaced in the left-hand side with \$ and all productions with only synchronizing nonterminals are given the ability to loop with \$ productions. In this way, nonsynchronized nonterminal productions are derived first, while the others can only loop. A similar trick is used to obtain a normal form for the restriction of SCF grammars in [2].

To make the construction of Lemma 11 slightly easier to read, we discuss a type of alternate normal form first. If I is the initial nonterminal of a BSCF grammar where I does not appear in the right-hand side of any production, then we can change I in all the productions to (I, b) and add the production $I \to (I, b)$. In addition, for all terminating productions $(A, x) \to \alpha$ where $\alpha \in T^*$ and $x \in \Xi$, we remove it if $x \in B$ as these could never be used in an accepting computation, we leave it if $x = e$, we change to $(A, x) \to (\lambda', e)$ and $(\lambda', e) \to \lambda$ if $x \in S$ and $\alpha = \lambda$ and we change to $(A, x) \to (a_1, e) \dots (a_n, e)$ and $(a_i, e) \to a_i$ if $x \in S$ and $\alpha = a_1 \cdots a_n$. Indeed, all derivations must start by going to the first nesting depth and all terminating productions must occur at the first nesting depth in an accepting computation. Consequently, we need not consider blocktrees and the inner tree separately. Furthermore, we can assume that all productions with a begin or end-nonterminal on the right-hand side must be of the form $(A, \lambda) \to (C, x)$ since we can replace all begin and end-nonterminals on the right-hand side of productions with an intermediate nonsynchronizing nonterminal which immediately must go to the original begin or end-nonterminal. In addition, for every occurrence of a terminal a on the right-hand side of a nonterminating production, we can replace it with (a', λ) and add $(a', \lambda) \to (a', e)$ and $(a', e) \to (a', e) \mid \lambda$. Thus, all productions are of the form either $I \to (I, b)$, $(A, x) \to (A_1, x_1) \cdots (A_n, x_n)$, $x \in \Xi \cup \{\lambda\}$, $n \geqslant 1$, $x_i \in S \cup \{\lambda\}$, $(A, \lambda) \to (B, y)$, $y \in B \cup \{e\}$ or $(A, e) \to w \in T^*$.

Lemma 11. *Let $G = (V, \Xi, T, I, P)$ be a* BSCF *grammar. Then we can construct a* BSCF *grammar $G' = (V', \Xi', T, I, P')$ without nonsynchronizing nonterminals (except for I) such that $L_z(G) = L_z(G')$ for $z \in \{\mathrm{p}, \mathrm{e}\}$.*

Proof. It suffices to consider the case where $z = \mathrm{p}$ since prefix synchronization can simulate equality without introducing any new nonsynchronizing nonterminals as shown in [10]. Assume without loss of generality that G keeps track of the synchronizing symbols on the base nonterminals (replace all occurrences of (V, x) in productions with (V_x, x)) and also that G is of the form discussed above. Let \$ be a new situation symbol and $\Xi' = S' \cup B \cup \{e\}$ where $S' = S \cup \{\$\}$. Define an intermediate production set P_1 by keeping all terminating productions in P, keeping $I \to (I, b)$ and also making the following changes: for all

productions in P of the form

$$(A_x, x) \to (A_{1_{x_1}}, x_1) \cdots (A_{n_{x_n}}, x_n), \text{ where } n \geqslant 1, x_i \in S \cup \{\lambda, e\}, \tag{8}$$

introduce

$$(A_x, \$) \to (A_{1_{x_1}}, \$) \cdots (A_{n_{x_n}}, \$) \tag{9}$$
$$\text{if } x = \lambda, x_i = \lambda \text{ for some } i, 1 \leqslant i \leqslant n,$$
$$(A_x, x) \to (A_{1_{x_1}}, x_1) \cdots (A_{n_{x_n}}, x_n) \mid (A_{1_{x_1}}, \$) \cdots (A_{n_{x_n}}, \$) \tag{10}$$
$$\text{if } x \neq \lambda, x_i \neq \lambda \text{ for all } i, 1 \leqslant i \leqslant n,$$
$$(A_x, \$) \to (A_{1_{x_1}}, x_1) \cdots (A_{n_{x_n}}, x_n) \mid (A_{1_{x_1}}, \$) \cdots (A_{n_{x_n}}, \$) \tag{11}$$
$$\text{if } x = \lambda, x_i \neq \lambda, \text{ for all } i, 1 \leqslant i \leqslant n,$$
$$(A_x, x) \to (A_{1_{x_1}}, \$) \cdots (A_{n_{x_n}}, \$) \tag{12}$$
$$\text{if } x \neq \lambda, x_i = \lambda \text{ for some } i, 0 \leqslant i \leqslant n.$$

Also, for all productions of the form $(A_\lambda, \lambda) \to (B_b, b)$, introduce

$$(A_x, \$) \to (B_b, b). \tag{13}$$

We define another intermediate production set P_2 by keeping all productions in P_1 and: for all productions in P_1 of the form $(A_x, x) \to \alpha$ such that $x \neq \lambda, x \in S \cup \{e\}, \alpha \in (N \cup T)^*$, $\alpha \notin T^*$ add

$$(A_x, \$) \to (A_x, x). \tag{14}$$

Define P' by keeping all productions in P_2 and: for all productions of the form $(A_x, \$) \to \alpha$ such that $\alpha \in (N \cup T)^*$, add

$$(A_x, \$) \to (A_x, \$). \tag{15}$$

"$\subseteq$" It is enough to consider the case for blocktrees. Let $t \in T_p(G)$ and let t_b be an arbitrary blocktree of t which does not contain any other blocktree. A blocktree t'_b of tree t' of G' is constructed exactly as in G until a nonsynchronizing nonterminal is encountered at the next height of the tree. Then, all nonterminals at that height are rewritten using the same productions as in G, except with \$ in all the nonterminals situation symbols, using productions created in (12) and the second part of (10). Then for the next heights, all the nonterminals that were originally synchronizing nonterminals loop using productions created in (15) until all other branches are done performing the nonsynchronized nonterminal productions. All of the nonterminals that originally were nonsynchronizing nonterminals carry out the same derivation recursively with \$ in place of λ using rules created in (9), until all nonsynchronizing nonterminals reach an end-nonterminal or go to a synchronizing nonterminal using rules created in the first part of (11). In this way, the nonsynchronizing nonterminal productions are derived first, while the others loop until all nonsynchronized nonterminal productions are finished. Then, the productions that were looping can proceed to what they were supposed to go to originally using rules created in (14), and the others can proceed to the next rule. This serves as a base case for induction. Let t_c be an arbitrary blocktree of t. We construct blocktree t'_c of t' exactly as in the base case except when we reach a production with a begin-nonterminal on the right-hand side. Then we use the rule created in

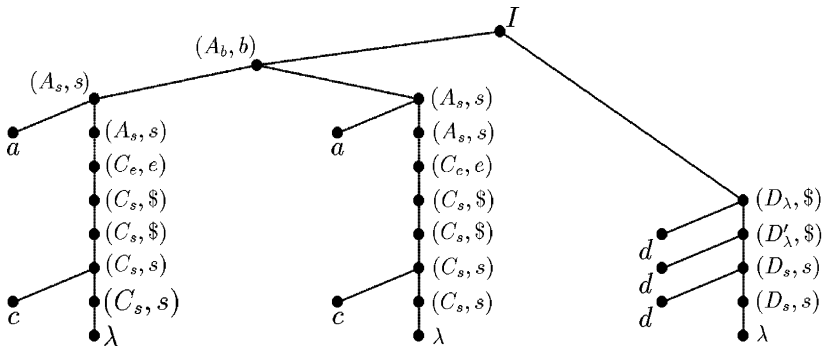

Fig. 2. A derivation tree $t' \in T_e(G')$ corresponding to the derivation tree $t \in T_e(G)$ in Fig. 1.

(13) whose blocktree can be constructed inductively. When that blocktree ends with rules from (10) and (12), we continue as in the base case. Thus, by induction, $\mathrm{yd}(t) = \mathrm{yd}(t')$ (see Fig. 2 for clarification).

"$\supseteq$" Let $t' \in T_p(G')$ and let t'_b be an arbitrary blocktree which does not contain any other blocktrees. The situation symbol must be the same for every height since there are no nonsynchronizing nonterminals. A blocktree t_b of tree t of G is constructed as in t'_b until the next height has \$ as its situation symbol. Then for each branch which applies a production created in the second part of (10) followed by a sequence of productions created in (15) and then (14), the original production in (8) will go directly to the next height without \$ as situation symbol. For each branch which applies a production created in (12), the original production in (8) will go directly to the nonterminals without λ as subscript at the next height without \$ as situation symbol. Furthermore, this continues when productions from (9) are applied. This process continues through all sections of \$ in the situation sequence. This serves as a base case for induction. Let t'_c be an arbitrary blocktree of t'. We construct a blocktree t_c of t exactly as in the base case until a begin nonterminal is reached. We construct this blocktree inductively until it ends and then continue as in the base case. Thus, by induction, $\mathrm{yd}(t') = \mathrm{yd}(t)$. Hence, we have $L_p(G) = L_p(G')$ and for any BSCF grammar, G, there exists a BSCF grammar, G' without nonsynchronizing nonterminals such that $L_z(G) = L_z(G')$ for $z \in \{\mathrm{p}, \mathrm{e}\}$. □

Example 12. Let G be as in Example 10. Although G uses equality synchronization and is not in the normal form used in Lemma 11, the idea is similar. We construct $G' = (V', \Xi', T, I, P')$, where $\Xi' = S' \cup B \cup \{e\}$, $S' = S \cup \{\$\}$, V' is obvious from construction and P' contains the following productions:

$I \to (A_b, b)(D_\lambda, \$)$, $\quad (A_b, b) \to (A_s, s)(A_s, s) \mid (A_s, \$)(A_s, \$)$,
$(C_e, e) \to (C_s, s) \mid (C_s, \$)$, $\quad (D_\lambda, \$) \to d(D'_\lambda, \$) \mid (D_\lambda, \$)$,
$(A_s, \$) \to (A_s, s) \mid (A_s, \$)$, $\quad (C_e, \$) \to (C_e, e) \mid (C_e, \$)$,
$(C_s, \$) \to (C_s, s) \mid (C_s, \$)$, $\quad (\lambda, \$) \to \lambda \mid (\lambda, \$)$,
$(A_s, s) \to a(A_s, s) \mid (C_e, e) \mid a(A_s, \$) \mid (C_e, \$)$,
$(C_s, s) \to c(C_s, s) \mid \lambda \mid c(C_s, \$) \mid (\lambda, \$)$,

$(D'_\lambda, \$) \to d(D_s, s) \mid d(D_s, \$) \mid (D'_\lambda, \$)$,
$(D_s, s) \to d(D_s, s) \mid \lambda \mid d(D_s, \$) \mid (\lambda, \$)$,
$(D_s, \$) \to (D_s, s) \mid (D_s, \$)$.

The tree t' has the \$ nonterminals where the nonsynchronizing nonterminals were, but it also "fills in" \$ nonterminals at the same place in all other branches of the same blocktree or of the inner tree, since accepting derivation trees of G' must also be e-synchronized. However, there are infinitely many possible e-acceptable derivation trees in $T_e(G')$ corresponding to t.

Note that $\mathrm{yd}(t) = \mathrm{yd}(t') = acacd^3$ and $L_e(G) = L_e(G') = \{a^n c^m a^n c^m d^{m+2} \mid n, m \geqslant 0\}$.

We note also that the above proof also works for strong derivations.

Eliminating the need for nonsynchronizing nonterminals is a very useful normal form as will be seen shortly. Here we state without proof a result from [10].

Theorem 13 (*Jürgensen and Salomaa [10]*). $\mathcal{L}_p(\mathrm{fnBSCF}) = \mathcal{L}_e(\mathrm{fnBSCF}) = \mathcal{L}_p(\mathrm{SCF}) = \mathcal{L}_e(\mathrm{SCF})$.

The following two results will be used to show the equivalence of finite nesting depth weak derivation languages to ET0L languages. This will help us determine some useful properties as ET0L systems have been studied extensively. For each production table of an ET0L system, we can associate a situation symbol. Whenever the ET0L system rewrites all nonterminals with a certain production table, an SCF grammar can rewrite all nonterminals with its corresponding situation symbol for all nonterminals. If a nonterminal uses a different situation symbol, the tree will no longer be equality synchronized. When all nonterminals are also terminals, this SCF grammar can terminate with the same terminals.

Lemma 14. *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}(\mathrm{ET0L}) \subseteq \mathcal{L}_z(\mathrm{SCF})$.

Proof. By Theorem 13, it is enough to show the result for $\mathcal{L}_e(\mathrm{SCF})$. Let $G = (V, V_1, \beta, w)$ be an ET0L system where $V_1 \subseteq V$, $V \neq \emptyset$, $w \in V^+$ is the initial nonterminal, and $\beta = \{P_1, \ldots, P_r\}$ is the set of tables of productions. Let $G' = (V \cup \{\lambda'\}, S, V_1, I, P)$ be a synchronized context-free grammar where $S = \{1, \ldots, r, \$\}$ (\$ is a new symbol), I is the starting nonterminal, V_1 is the set of terminals, and V, the set of base nonterminals (except for λ'), is the same as the set of nonterminals [1] of G. Define P as follows:

If $w = a_1 \cdots a_k$, where $a_1, \ldots, a_k \in V$, with $k \geqslant 1$, then let
  $I \to (a_1, x) \cdots (a_k, x) \in P$ for all $x \in S$.
For each P_i, $1 \leqslant i \leqslant r$, for each $a \in V$, if $a \to w_a \in P_i$
with $w_a = c_1 \cdots c_l$, $c_m \in V$, $1 \leqslant m \leqslant l$ then
  $(a, i) \to (c_1, x) \cdots (c_l, x) \in P$ $\forall x \in S$ and if $w_a = \lambda$, then let
  $(a, i) \to (\lambda', x) \in P$, $\forall x \in S$.

[1] Although the set of base nonterminals has a nonempty intersection with the terminals, the set of nonterminals, which are ordered pairs, has an empty intersection with the terminals.

Also, let $(\lambda', x) \to (\lambda', y)$, $\forall x, y \in S$.
For all $a \in V_1$, associate $(a, \$) \to a \in P$ and also $(\lambda', \$) \to \lambda \in P$.

Given an accepting derivation $w \Rightarrow_G^* v \in V_1^*$, G' simulates this derivation at each step by rewriting all nonterminals at a given height using the situation symbol that corresponds with the production table index. If a nonterminal in G goes to λ, then G' goes to a nonterminal (λ', j), for some j, which carries out the derivation using the same situation symbol the other branches are using. Once G gets to v, all the symbols in v can go to the corresponding terminals, or if $v = \lambda$, then all branches can go to λ.

Given an accepting derivation tree in G', notice by construction that all situation sequences must end in \$ and be of the same length, and that there are no nonsynchronizing nonterminals. So, given any height of the tree, the situation symbol for every branch at that height is identical. This means that at each height, each nonterminal has situation symbol i, say, and by construction all productions are in production table i or have already gone to λ. The tree must end with terminals, hence by construction, the terminals must all belong to V_1.

Thus, $L(G) = L_e(G')$ and $\mathcal{L}(\text{ET0L}) \subseteq \mathcal{L}_z(\text{SCF})$. □

The reverse inclusion is similar, associating a production table with every situation symbol; the key being the ability to assume without loss of generality that there are no nonsynchronizing nonterminals. Notice in the proof of Lemma 14, that no nonsynchronizing nonterminals were introduced. If the SCF grammar uses a given situation symbol for all nonterminals at a certain height (this must be the same since there are no nonsynchronizing nonterminals), then the ET0L system can rewrite all variables with the corresponding production table. Since there must be a production for each nonterminal in each production table, we make productions so that if a nonterminal has a certain situation symbol, it can only get rewritten to a "dead nonterminal" by another production table.

Lemma 15. *For* $z \in \{\text{p}, \text{e}\}$, $\mathcal{L}_z(\text{SCF}) \subseteq \mathcal{L}(\text{ET0L})$.

Proof. By Theorem 13, it is enough to show the result for $z = \text{p}$. Without loss of generality (by Lemma 11), let $G = (V, S, T, I, P)$ be a synchronized context-free grammar without nonsynchronizing nonterminals. Also, I can be considered a synchronized nonterminal which has its own situation symbol and does not appear on the right-hand side of any production. Let $S = \{1, \ldots, r\}$. Let $G' = (\bar{V}, \bar{V}_1, \beta, I)$ be an ET0L system where $\bar{V} = N \cup T \cup \{u\}$ (u is a new symbol) is the set of nonterminals, $\bar{V}_1 = T$ is the set of terminals, and $\beta = \{P_1, \ldots, P_r\}$ is the system of production tables. The productions are defined as follows:

$u \to u \in P_x$ $\forall x$,
$t \to t \in P_x$, $\forall t \in T$, $\forall x$,
If $(A, x) \to w \in P$, then
$(A, x) \to w \in P_x$ and $(A, x) \to u \in P_y$, $\forall y \neq x$.

We can think of u as a "dead nonterminal", meaning if we ever get a u in a derivation, it should never accept.

Given a p-acceptable derivation tree, since there are no nonsynchronizing nonterminals, then at a given height all the nonterminals left at that height must have the same situation symbol, x, say. All of these nonterminals get rewritten to either terminals or nonterminals with situation symbol y, say. By construction, all these productions are in P_x and all terminals that have already been reached go to themselves. This continues until all the branches reach terminals, and then the derivation can accept in G'.

Given an accepted derivation $I \Rightarrow^*_{G'} w \in \bar{V}_1^*$ of G', notice that since $w \in \bar{V}_1^*$, u is not a letter of w. This means that the only productions used in G' are of the form $(A, x) \to \alpha \in P_x$. At each step, each nonterminal is rewritten using a production table P_x, say. This means that at each height, all nonterminals being rewritten at that height in the derivation for G are using nonterminals with situation symbol x or have already reached terminals. Thus, the situation symbols will be the same at a given height of the derivation tree for G. Once it reaches w, the derivation has reached terminals w in G by construction, and will accept.

Hence, $L_{\mathrm{p}}(G) = L(G')$ and $\mathcal{L}_z(\mathrm{SCF}) \subseteq \mathcal{L}(\mathrm{ET0L})$. □

Combining Lemmas 14, 15 and Theorem 13 above, we have

Proposition 16. *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}_z(\mathrm{SCF}) = \mathcal{L}_z(\mathrm{fnBSCF}) = \mathcal{L}(\mathrm{ET0L})$.

This leads us to some well known facts about ET0L languages.

Corollary 17. *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}_z(\mathrm{fnBSCF})$ *is a Full AFL.*

It is known that every ET0L language can be generated by a propagating (λ-free) ET0L system [13], and the construction in Lemma 14 does not introduce any λ-productions from any nonerasing productions.

Corollary 18. *Let G be a BSCF grammar. If* $L_z(G) \in \mathcal{L}_z(\mathrm{fnBSCF})$, *we can construct a BSCF grammar* G' *such that* G' is λ*-free and* $L_z(G) - \{\lambda\} = L_z(G')$ for $z \in \{\mathrm{p}, \mathrm{e}\}$.

As noted in [10], SCF languages are exactly the yields of nondeterministic synchronized tree automata [14].

Corollary 19. *The family of string languages generated by both nondeterministic prefix and equality synchronized tree automata coincides with* $\mathcal{L}(\mathrm{ET0L})$.

5. Unbounded nesting depth weak derivations

Up to this point, we are unsure of whether the weak derivation languages (not necessarily bounded nesting depth) will be strictly larger than finite nesting depth weak derivation languages as conjectured in [10]. The following two lemmas will show that this is exactly the case. We state without proof:

Lemma 20 (*Jürgensen and Salomaa [10]*). *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}_z(\mathrm{BSCF}) \subseteq \mathcal{L}(\mathrm{IND})$.

To see the reverse inclusion, we notice that blocks are similar to indices in that they are in stack form. We can keep track of the top index on base nonterminals. The largest problem is that when we end a block, which corresponds with popping an index, the derivation is unaware of the new top stack symbol. To solve this problem, whenever we enter a block, we use a situation symbol that is associated with the index in one branch and continue the derivation in another branch. In this way we can nondeterministically calculate the top index if we re-enter the block.

Lemma 21. $\mathcal{L}(\mathrm{IND}) \subseteq \mathcal{L}_{\mathrm{p}}(\mathrm{BSCF})$.

Proof. Without loss of generality, let $G = (V', T, I, F, P)$ be an indexed grammar in normal form. Let $F = \{f_1, \ldots, f_n\}$ be the set of indices, V' be the set of nonterminals, I be the starting nonterminal, and P be the set of productions. Let $G' = (V, \Xi, T, I', P')$ be a BSCF grammar where $\Xi = S \cup B \cup \{e\}$, $S = \{s_0, \ldots, s_n\}$, and $B = \{b\}$ with n as above. The set of nonterminals N of G', will be obvious from the construction.

If a blocktree is being derived at a given nesting depth x, the derivation can be thought of as performing a derivation with the xth stack symbol being at the top.

For all productions in P of the word $A \to CD$, $A, C, D \in V'$, let
$(A_j, \lambda) \to (C_j, \lambda)(D_j, \lambda) \in P'$ for all j, $0 \leqslant j \leqslant n$.

The j's can be thought of as the top index symbol (0 being no index). So, this form of production can occur no matter what the top index is, and j is the same top symbol for C and D. Indexed grammars are only aware of the top index for any given production. Productions of this form in G will give C and D the same stack of indices as A.

For all productions in P of the form $A \to a$, $A \in V'$, $a \in T \cup \{\lambda\}$, let
$(A_0, \lambda) \to a \in P'$,
$(A_j, \lambda) \to (a', e) \in P'\ \forall j, 1 \leqslant j \leqslant n$,
$(a', e) \to (a', e) \mid a \in P'$ where a' is a new symbol in bijective correspondence with a.

Whenever A goes to a terminal in G, it "forgets" all its indices. For it to "forget" all its indices in G', it must go to nesting depth 0 and then go to the corresponding terminal to remain well-formed.

For all productions in P of form $A \to Cf_i$, $A, C \in V'$, $f_i \in F$, let
$(A_j, \lambda) \to (C_i, b) \in P'\ \forall j, 0 \leqslant j \leqslant n$,
$(C_i, b) \to (C_i, s_i)(C_i, \lambda) \in P'$,
$(C_i, s_i) \to \lambda \mid (\lambda', e) \in P'$,
$(\lambda', e) \to (\lambda', e) \mid \lambda \in P'$.

This starts a new blocktree with top index f_i and then starts two branches, one that uses s_i; the corresponding situation-symbol (so we can later determine f_i is the top index when we re-enter the block) and then ends, so that it is well-formed and does not change the yield, and one that continues the derivation using nonsynchronizing nonterminals.

For all productions in $f \in F$ of form $A \to C$, $A, C \in V'$, let
$(A_f, \lambda) \to (C_f, e) \in P'$,
$(C_f, e) \to (C_0, s_0)(C_0, \lambda) \mid \cdots \mid (C_n, s_n)(C_n, \lambda) \in P'$,
$(C_j, s_j) \to (\lambda', e) \mid \lambda \in P', \forall j, 0 \leqslant j \leqslant n$,
$(\lambda', e) \to (\lambda', e) \mid \lambda \in P'$.

We know the top index is *f*, so we end and we now have a new unknown top index symbol. We can nondeterministically calculate the top index symbol since we used an s_i representing the top symbol when we entered the blocktree. Then we continue the derivation on another branch with the first branch ending without changing the yield.

Also, let $I' \to (I, s_0)(I, \lambda) \in P'$.

It is clear from the descriptions that $L(G) = L_{\mathrm{p}}(G')$.

Hence $\mathcal{L}(\mathrm{IND}) \subseteq \mathcal{L}_{\mathrm{p}}(\mathrm{BSCF})$. □

Notice in the proof of Lemma 21 that given any $t \in T(G')$, where t_1 is a blocktree of t or $t_1 = \mathrm{inner}(t)$, we have for all $\mu \in \mathrm{leaf}(t_1)$, $\mid \mathrm{sit}_{t_1}(\mu) \mid = 0$ or 1. This is true since we always use a situation symbol when we enter a new blocktree and then potentially again when we enter back into the same nesting depth in the branch to the right of the first one with the first one ending and so on. So we can easily modify the proof of Lemma 21 so that $\mid \mathrm{sit}_{t_1}(\mu) \mid = 1$ always and hence $\mathcal{L}(\mathrm{IND}) \subseteq \mathcal{L}_{\mathrm{e}}(\mathrm{BSCF})$. This gives an alternate proof (the proof given in [11] works for both weak and strong derivations) that $\mathcal{L}_{\mathrm{e}}(\mathrm{BSCF}) = \mathcal{L}_{\mathrm{p}}(\mathrm{BSCF})$. Thus, we obtain:

Proposition 22. *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}(\mathrm{IND}) = \mathcal{L}_z(\mathrm{BSCF})$.

The following is seen directly from well known facts about indexed languages:

Corollary 23. *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}_z(\mathrm{BSCF})$ *is a Full AFL.*

The next result is a direct consequence of Propositions 16 and 22, and the fact that the family of ET0L languages is a proper subset of the family of indexed languages (see [5]).

Theorem 24. *For* $z \in \{\mathrm{p}, \mathrm{e}\}$, $\mathcal{L}_z(\mathrm{fnBSCF}) \subset \mathcal{L}_z(\mathrm{BSCF})$.

6. Conclusions

We have given an alternate characterization for the important family of ET0L languages as being equal to both the SCF languages and the BSCF languages where the nesting or recursion depth is bounded above by any constant using equality or prefix synchronization. This is useful for the study of the ET0L languages as it is often easier to describe a language using a BSCF grammar. Indeed, one can use various methods of synchronization, nonsynchronizing nonterminals to hide events that need not synchronize and any amount of recursion, so long as the depth of recursion is bounded above by a constant. In addition, they often simplify proofs. Recently, synchronized context-free grammars were used to show that the family of ET0L languages is closed under the hi (hairpin inversion) and the

dlad (double loop with alternating direct pointers) biooperations, making use of nonsynchronizing nonterminals in the constructions [3]. The characterization is also useful for the study of nondeterministic synchronizing tree automata, as it shows that the language family obtained by the yields of the trees is equal to the family of ET0L languages.

Moreover, we showed that when the recursion of BSCF grammars can get arbitrarily deep, the generated language families using both synchronization types are equal to the family of indexed languages. This allows one to use both recursion and synchronization when describing indexed languages. This is often significantly easier and more intuitive. Furthermore, it demonstrates that BSCF languages are strictly more general than finite nesting depth BSCF languages using both types of synchronization, as conjectured by Jürgensen and Salomaa in [10].

Acknowledgements

We thank Professor Helmut Jürgensen for his help.

References

[1] A. Aho, Indexed grammars—an extension of context-free grammars, J. ACM 15 (1968) 647–671.
[2] H. Bordihn, M. Holzer, On the computational complexity of synchronized context-free languages, J. Universal Comput. Sci. 8 (2) (2002) 119–140.
[3] M. Daley, O. Ibarra, L. Kari, I. McQuillan, K. Nakano, Closure and decision properties of some language classes under ld and dlad bio-operations, J. Autom., Lang. and Combin. 8 (2003) 477–498.
[4] J. Dassow, G. Păun, Regulated Rewriting in Formal Language Theory, EATCS Monographs in Theoretical Computer Science, Vol. 18, Springer, Berlin, 1989.
[5] J. Engelfriet, E. Schmidt, J. van Leeuwen, Stack machines and classes of nonnested macro languages, J. ACM 27 (1) (1980) 97–117.
[6] J. Hopcroft, J. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[7] J. Hromkovič, How to organize the communication among parallel processes in alternating computations, Unpublished manuscript, Comenius University, Bratislava, 1986.
[8] J. Hromkovič, J. Karhumäki, B. Rovan, A. Slobodová, On the power of synchronization in parallel computations, Discrete Appl. Math. 32 (1991) 155–182.
[9] J. Hromkovič, B. Rovan, A. Slobodová, Deterministic versus nondeterministic space in terms of synchronized alternating machines, Theoret. Comput. Sci. 132 (1994) 319–336.
[10] H. Jürgensen, K. Salomaa, Block-synchronization context-free grammars, in: Z. Du, I. Ko (Eds.), Advances in Algorithms, Languages, and Complexity, Kluwer Academic Publishers, The Netherlands, 1997, pp. 111–137.
[11] I. McQuillan, Descriptional complexity of block-synchronization context-free grammars, in: J. Dassow, M. Hoeberechts, J. Jürgensen, D. Wotschke (Eds.), Descriptional Complexity of Formal Systems (DCFS), Pre-Proc. of a Workshop, Department of Computer Science, University of Western Ontario, London, Canada, 2002, pp. 188–203 (Report No. 586).
[12] I. McQuillan, New results involving block-synchronization context-free grammars, Technical Report No. 258, Department of Computer Science, University of Western Ontario, a previous version of this manuscript, 2002.
[13] G. Rozenberg, A. Salomaa, The Mathematical Theory of L Systems, Academic Press, Inc., New York, 1980.
[14] K. Salomaa, Synchronized tree automata, Theoret. Comput. Sci. 127 (1994) 25–51.